VH2FG
EIN VHDL NACH FLUSSGRAPH KONVERTER FÜR CADDY

M. SCHMIDT[1], U. MÖHRKE[2], P. HERRMANN[3]

Institut für Informatik
Universität Leipzig, Augustusplatz 10-11
04103 Leipzig
Germany

Universität Leipzig / Institut für Informatik
Report Nr. 4 (1996)

[1]email: schmidt@peanuts.informatik.uni-tuebingen.de
[2]email: moehrke@regent.e-technik.tu-muenchen.de
[3]email: paul@hpws1.informatik.uni-leipzig.de

Zusammenfassung

Für das Erstellen von Schaltungen wird immer mehr die Hardwarebeschreibungssprache VHDL eingesetzt. Sie bietet viele Vorteile gegenüber den bisherigen Beschreibungsmethoden, wie zum Beispiel das Aufbauen der Schaltung mit Hilfe von graphischen CAD Werkzeugen. Ein wichtiger Faktor ist dabei die Entwicklungszeit. Dazu benötigt man aber auch leistungsfähige Syntheseprogramme. Das High–Level Syntheseprogramm CADDY ist nicht in der Lage direkt VHDL-Quelltext zu verarbeiten. Mit dem Programm `vh2fg` existiert nun die Möglichkeit die VHDL-Beschreibung in ein Flussgraphenformat umzuwandeln, das CADDY verarbeiten kann.

Hier wird eine Anleitung gegeben, wie man mit diesem Programm umgeht. Es wird die VHDL–Eingabe spezifiziert und die Ausgabe in Flussgraph erklärt.

1 Einleitung

Dieser Bericht entstand aus der Arbeit an einem Projekt names „FPGA-Entwurf“, das von der DFG gefördert wird. In diesem Projekt arbeiten Lehrstühle von drei Universitäten zusammen: *Lehrstuhl für Rechnersysteme der Universität Leipzig*[4], *Lehrstuhl für Rechnergestütztes Entwerfen der TU München*[5] und *Lehrstuhl für Technische Informatik der Universität Tübingen*[6].

Nach der Bestimmung des aktuellen Standes der Forschung und des Design Flow Management, welches in [SMH95] dargelegt ist, sollten die Lücken in der Formatkonvertierung geschlossen werden. Das grösste Paket in diesem Bereich war die Konvertierung von VHDL [IEE88] [IEE93] nach Flussgraph [CW85].

Von Flussgraph aus kann eine High–Level Synthese mit dem Syntheseprogramm CADDY [CR89] durchgeführt werden. CADDY wandelt die Verhaltensbeschreibung auf algorithmischer Ebene in eine Register–Transfer Beschreibung um. Anschliessend kann eine bibliotheksorientierte Technologieabbildung [KKT+95] durchgeführt werden.

Die Abbildung von hier auf das FPGA[7] wird auch in diesem Projekt bearbeitet. Die Platzierung der einzelnen CLBs[8] wird in München vorgenommen. An der Verdrahtung der CLBs in dem FPGA wird in Leipzig gearbeitet. Diese beiden Arbeitsgebiete sind Teil dieses Projekts.

2 `vh2fg`

Mit diesem Programm lassen sich VHDL-Verhaltensbeschreibungen in einen Flussgraphen umwandeln, der vom Syntheseprogramm CADDY verarbeitet werden kann. Die Umwandlung soll exakt die VHDL-Beschreibung in Flussgraphdarstellung ausdrücken, möglichst nicht mehr und nicht weniger. Dieses kann nicht immer gewährleistet werden, da VHDL mächtiger ist als der Flussgraph.

2.1 Installation

Das Programm `vh2fg` ist in Standard C geschrieben und lässt sich auf allen Unix–Plattformen übersetzen. Getestet wurde es auf SunOS 4, Solaris 2.5,

[4] Prof. Dr.-Ing. Spruth
[5] Prof. Dr.-Ing. Antreich
[6] Prof. Dr. rer. nat. Rosenstiel und Prof. Dr. rer. nat. Kebschull
[7] Field Programmable Gate Array
[8] Configurable Logic Blocks

Linux und AIX 3.

2.2 Benutzung

Das Programm hat vier Parameter. Zwei davon sind dazu bestimmt die Eingabe und die Ausgabe in eine Datei umzuleiten. Mit `-in file` gibt man das Eingabefile an, welches die VHDL-Beschreibung enthält und mit `-out file` die Flussgraph–Ausgabedatei. Wird keine Datei angegeben, kommt die Eingabe von der Standardeingabe und die Ausgabe geht in die Standardausgabe. Fehler, Warnungen und sonstige Kontrollausgaben des Programms werden auf dem Standardfehlerkanal ausgegeben.

Mit einem weiteren Parameter `-clk signal` kann ein Signal aus dem ENTITY angegeben werden, welches als Takt spezifiziert wird. Im allgemeinen ist das nicht nötig, wurde aber der Vollständigkeit halber dazugenommen.

Der letzte Parameter `-d` dient zur Fehlersuche und steht nur zur Verfügung, wenn das Programm mit dem Makro `DEBUG` übersetzt wurde. In diesem Fall, wird damit die Debug-Ausgabe vom Parser eingeschaltet. Damit lassen sich Parse-Fehler sehr gut lokalisieren.

```
usage: vh2fg [-in infile] [-out outfile] [-clk clocksignal] [-d]
```

3 Eingabe VHDL

VHDL ist eine sehr mächtige Sprache zur Erstellung von Schaltungen. Das Besondere daran ist, dass man sie auf jeder Abstraktionsebene zur Beschreibung einsetzen kann. Damit verbunden ist der Nachteil, dass es Konstrukte gibt, die zwar in VHDL durchaus erlaubt sind, aber in der entsprechenden Abstraktionsebene keine Sinn manchen. In diesem Fall sind wir in der Verhaltensbeschreibung auf Algorithmischer Ebene (Abbildung ??) [SMH96].

Hier sind nun die Konstrukte, die das `vh2fg` Programm verarbeiten kann. Es ist nur eine Aufzählung, denn sie können so wie es in VHDL definiert ist eingesetzt werden. Siehe dazu die Referenzen zu VHDL [IEE88] [IEE93] oder andere Lehrbücher zu VHDL [Leh94].

In diesem Kapitel soll nur beschrieben werden, welche Teile von VHDL von `vh2fg` verarbeitet werden. Die korrespondierenden Flussgraph–Anweisungen werden im Kapitel 4 beschrieben.

3.1 Datentypen

Bei den Datentypen ist man vom Flussgraphen her sehr eingeschränkt. Sicherlich wird sich das mit zukünftigen Arbeiten ändern. Der Flussgraph lässt

keine komplexen Datentypen zu. In der jetztigen Implementierung sind erlaubt:

- `BIT`
- `BIT_VECTOR`
- `STD_LOGIC_VECTOR`
- `SIGNED`
- `UNSIGNED`
- `INTEGER`

Die Umsetzung im Flussgraph ist aber immer auf ganze Zahlen ohne Vorzeichen. Diese Skalaren Datentypen können noch mit

```
TYPE name IS ARRAY (x DOWNTO y) OF BIT_VECTOR(a DOWNTO b);
```

als Feld definiert werden.

Da es in der Verhaltensbeschreibung keine Signale gibt ist nur die Definition als Variable erlaubt.

Auch die Konstantendefinition ist erlaubt.

3.2 Interface

Das Interface wird wie in VHDL üblich als `ENTITY` beschrieben. Als Signalrichtungen gibt es `IN, OUT` und `INOUT`. Weiterhin müssen die Beschränkungen für Datentypen beachtet werden.

3.3 Blöcke

Die Beschreibung besteht immer aus einer `ARCHITECTURE`. Darin kann ein `PROCESS` enthalten sein, muss aber nicht. Da aber Variablen nur in Prozesskopf definiert werden dürfen, wird er häufig benötigt. Eine eventuell angegebene *sensitivity list* wird ignoriert.

3.4 Zuweisungen

Bei Zuweisungen gibt es keine Einschränkungen.

3.5 Kontrollstrukturen

Die zwei gebräuchlichsten Kontrollstrukturen existieren auch in der Flussgraphbeschreibung. Dies sind `IF THEN ELSE` und die `CASE` Anweisung.

3.6 Schleifen

Es sind 3 Schleifentypen implementiert.

- `FOR IN LOOP`
- `WHILE LOOP`
- `LOOP ... EXIT WHEN ... END LOOP`

Dazu kommt noch eine Pseudo-Schleife namens `WAIT UNTIL`.

Die Letzte der drei Schleifentypen beschreibt eine *UNTIL* Schleife. Diese ist in Flussgraph vorhanden aber in VHDL nur über den Umweg der `LOOP` Anweisung möglich. Dabei muss die `EXIT` Anweisung am Ende des Schleifenrumpfes stehen.

3.7 WAIT–Anweisungen

In der Verhaltensbeschreibung ist es fraglich, ob WAIT–Anweisungen überhaupt Sinn machen. Im vorigen Paragraphen wurde schon das `WAIT UNTIL` erwähnt. Diese Anweisung wird einfach in eine äquivalente WHILE-Schleife umgewandelt.

`WAIT UNTIL x;` $\Leftrightarrow$ `WHILE not x LOOP ENDLOOP;`

Die zweite WAIT–Anweisung die implementiert wurde ist das `WAIT FOR time`. Damit wartet man eine gewisse Zeit. Die Ausführung wird möglicherweise über mehere Takte hinweg unterbrochen. Siehe dazu [GR94].

3.8 Komponenten

Komponenten bilden die Schnittstellen zwischen der Verhaltensbeschriebung und der Register–Transfer Struktur externer Bausteine. Dazu wird die Komponentendeklaration direkt in die VHDL Datei geschrieben. Sie bildet die Schnittstelle. Auf die Komponete wird dann mit Hilfe von Prozeduren zugegriffen. Um die Prozeduren den Komponenten zuordnen zu können müssen die Namen der Prozeduren mit dem Komponentennamen beginnen. Beispiel:

```
COMPONENT ram PORT ( .... ) END COMPONENT ;
```

dann müssen die Prozeduren folgendermassen heissen:
`ram_read, ram_write, ...`
Von den Prozeduren muss nur die Deklaration existieren. Die Rümpfe werden überlesen und werden auch nicht mitsynthetisiert, denn sie werden in Register–Transfer beschrieben. Behandelt werden diese Prozeduren wie die alle anderen Funktionen und Operatoren, zum Beispiel `+, -, *, ...`

Diese Art Komponenten zu beschreiben ist nicht ganz VHDL-Standard. Normalerweise werden sie im `PACKAGE` beschrieben und per `LIBRARY`–Aufruf eingelesen.

3.9 Beispiel

Als Beispiel bietet sich eine kurze, jedem bekannte, mathematische Funktion an. Die Funktion zur Bestimmung des grössten gemeinsamen Teilers (ggT) ist so eine.

```
 1  ENTITY ggt IS
 2    PORT( I1 : IN BIT_VECTOR(15 DOWNTO 0);
 3          I2 : IN BIT_VECTOR(15 DOWNTO 0);
 4          OU : OUT BIT_VECTOR(15 DOWNTO 0));
 5  END ggt;
 6
 7
 8  ARCHITECTURE arch1 OF ggt IS
 9
10  BEGIN
11    PROCESS
12      VARIABLE X1, X2 : BIT_VECTOR(15 DOWNTO 0);
13    BEGIN
14      X1 := I1;
15      X2 := I2;
16      WHILE X1 /= X2 LOOP
17        IF X1 < X2 THEN
18          X2 := X2 - X1;
19        ELSE
20          X1 := X1 - X2;
21        END IF;
22      END LOOP;
23      OU <= X1;
24    END PROCESS;
25  END ;
```

Die Zeilennummern gehören nicht zur VHDL-Beschreibung, sie dienen nur der Übersicht und zur Referenz.

4 Ausgabe Flussgraph

Die Flussgraph-Beschreibung wird in [CW85] genau spezifiziert. Aber CADDY kann nicht alles was dort definiert ist verarbeiten. Auf genau diese Unterschiede wird hier eingegangen. Ausserdem wird beschrieben, wie die VHDL–Anweisungen in Flussgraph umgesetzt werden.

4.1 Datentypen

In der Definition sind sechs Skalare Datentypen spezifiziert, das sind:

- Ganzzahlig ohne Vorzeichen — `LOG`
- Ganze Zahl im Zweierkomplement — `TWC`
- Ganze Zahl im Einerkomplement — `ONC`
- „Binary Coded Decimal“ – Darstellung — `BNC`
- Festpunktdarstellung — `FIX`
- Gleitpunktdarstellung — `FLT`

CADDY verarbeitet davon nur den `LOG`–Datentyp. Die Bitbreite der Variablen von Typ `LOG` müssen in der Deklaration angegeben werden. Arrays sind auch erlaubt.

Im Vergleich zu VHDL ist das nur sehr wenig. Allerdings kann man darauf theoretisch alle Datentypen abbilden. Man kennt das von der Programmierung mit höheren Programmiersprachen. Es können abstrakte Datentypen definiert werden und der Compiler muss diese in ein lineares Feld von Speicherzellen übersetzen.

4.2 Interface

Neben den Informationen, die man aus dem `ENTITY` der VHDL–Beschreibung hat, stehen hier noch einige Parameter, die die elektrischen Eigenschaften der Hardware beschreiben. Diese sind aber nicht relevant und werden auch nicht weiter beachtet.

4.3 Blöcke

Der wichtigste Block ist die `FUNCTIONAL`–Beschreibung. Dies ist das Äquivalent zum `ARCHITECTURE, PROCESS`. Daneben gibt es noch den `MODULE`–Block, dieser wird für die Komponenten benötigt und dort beschrieben.

4.4 Zuweisungen

Im Flussgraph gibt es nur Drei–Adress–Zuweisungen. Das bedeutet, dass der Übersetzer die komplexen Zuweisungen in diese umwandeln muss. Dazu ist auf die Prioritäten der Operationen zu achten. Alle Operationen die es in VHDL gibt, gibt es auch in der Flussgraphbeschreibung.

4.5 Kontrollstrukturen

Die `IF THEN ELSE` Anweisungen können nahezu eins zu eins in Flussgraph übersetzt werden. Bei den Bedingungen muss man wie bei den Zuweisungen vorgehen. Für die `CASE`–Anweisung gilt Entsprechendes.

4.6 Schleifen

Auch Schleifen sind in Flussgraph vorhanden. Für sie gilt das gleiche wie für Konrollstrukturen.

4.7 WAIT–Anweisung

Es wird nur eine WAIT–Anweisung unterstützt. Das ist die `WAIT FOR time` in VHDL. Näheres hierzu siehe [GR94]. Der Zeitabschnitt kann in VHDL in verschiedenen Grössen angegeben werden. Für den Flussgraph werden sie in Caddy–Sekunden (cs) umgerechnet Eine Caddy–Sekunde ist $\frac{1}{100}$ Nano–Sekunde.

4.8 Komponenten

Jede in VHDL-Beschriebene Komponente wird zu einem `MODULE` zusammengefasst. Das `MODULE` besteht aus der Deklaration des `INTERFACE` und den `PERFORMED FUNCTIONS`. Im Interface werden die Ein/Ausgabe Signale der Komponente aufgezählt. Dazu kommt noch die Deklaration der Funktionen mit deren Ein/Ausgängen. Mit dieser Deklaration können die Funktionen im Flussgraphen genauso verwendet werden wie die üblichen vordefinierten Operatoren (`+, -, *,...`).

Die genaue Spezifikation der Funktionen steht in einer Deklarationsdatei von CADDY (`komp_lib.sys`). In dieser Datei stehen die elektrischen Daten wie die Grösse, die Anzahl der Gatter, das Verhalten der Ausgänge zu den Eingängen.

4.9 Beispiel

Den kompletten GGT hier in Flussgraphendarstellung darzustellen, ist etwas zu viel. Dafür ist in Abbildung 1 der Flussgraph als Diagramm zu sehen.

Dennoch soll am Beispiel des ersten Knotens aus dem GGT der Aufbau kurz erleutert werden.

```
Index       : 1      Tag : SO      Line : 14      Column : 2
Operation   : TR
Inputs      : I1(15..0) : IN
Outputs     : X1(15..0) : LOG
Predecessor: *
Condition   : X     Successor : 2
```

Dieser Knoten korrespondiert zu Zeile 14 in der VHDL–Beschreibung. Die Felder bedeuten im einzelnen:

Index: Das ist die Nummer des Knotens.

Tag: bezeichnet die Kontrollstruktur zu der die Operation gehört. In diesem Fall ist es `SO`, was *Single Operation* bedeutet. Damit werden zum Beispiel auch der Anfang und das Ende einer Schleife bezeichnet (`WTS` = *While Test Single*, `WE` = *While End*).

Line: Hier steht die Zeilennummer der korrespondierenden VHDL–Beschreibung.

Column: Damit wird die Position innerhalb der Zeile angegeben. Sowohl `Line` als auch `Column` können beliebige Werte enthalten und werden nur für die Fehlersuche verwendet.

Operation: ist die Funktion

Inputs: die Eingänge

Outputs: die Ausgänge

Predecessor: die Liste der Vorgängerknoten

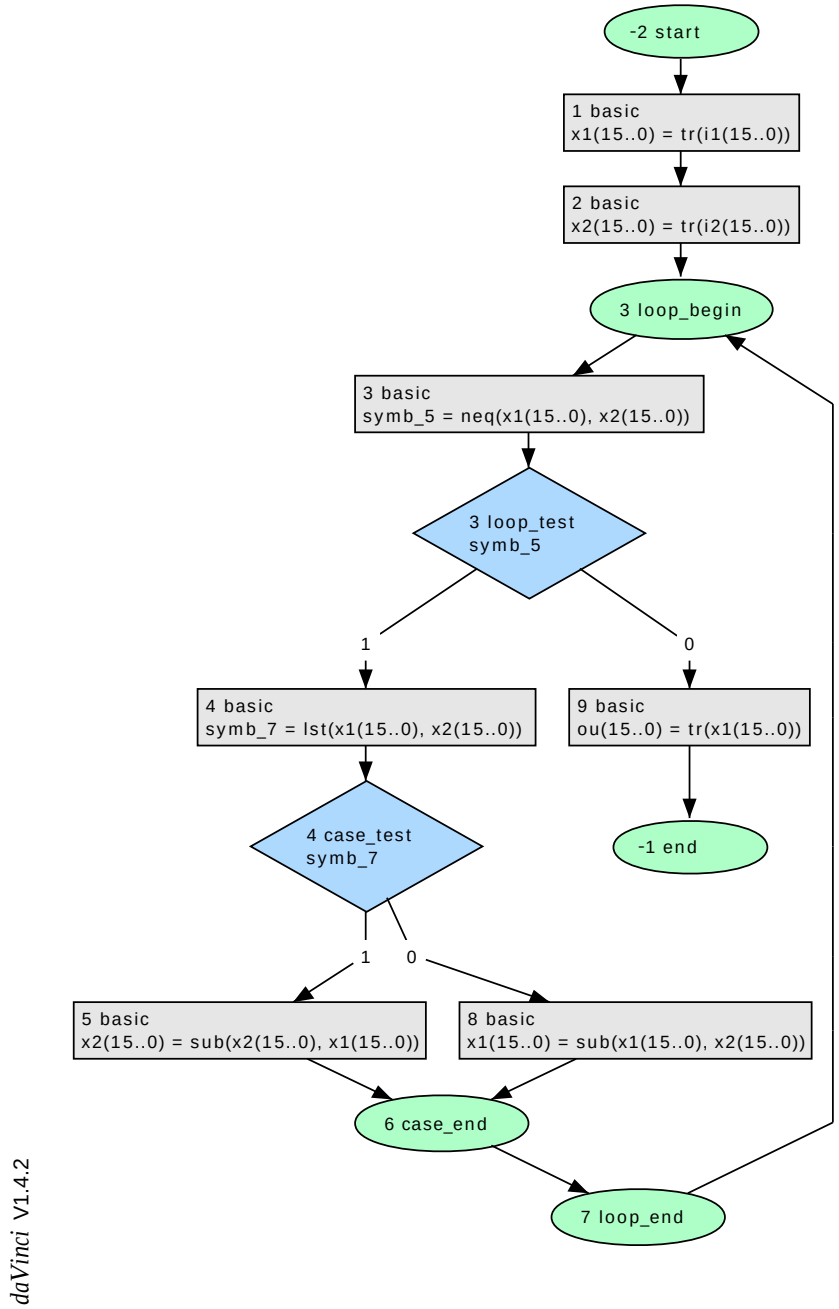

Abbildung 1: GGT als Flussgraph

Condition: Bei Verzweigungen wie z.B. bei `IF THEN ELSE` steht hier die Bedingung die der Ausgang zum Verzweigen haben soll.

Successor: ist der Nachfolgerknoten. Bei Verzweigungen stehen mehrere `Condition` und `Successor` untereinander.

5 Ausblick

Zum Testen von CADDY und um neue Beispiele zu rechnen ist dieses Programm sicherlich ausreichend. Für komplexere Schaltungen, vor allem wenn abstrakte Datentypen verwendet werden sollen, ist dieses Programm nicht ausgelegt. Gerade im Bereich Datentypen lässt sich noch einiges verbessern und ausbauen. Sicherlich wird das das Thema von verschiedenen zukünftigen Arbeiten sein.

Das wichtigste ist, dass die Eingabe für CADDY nun etwas einfacher gestaltet ist. Und neue Beispiele können ohne grössere Probleme erstellt werden. Dies hilft auch der Weiterentwicklung von CADDY. Damit verbunden sind natürlich auch alle Programme, die mit den CADDY–Ausgaben weiterarbeiten. Das wird sich bis auf die unterste Ebene der Synthese auswirken.

Literatur

[CR89] Raul Camposano and Wolfgang Rosenstiel. Synthesizing Circuits From Behavioral Description. *IEEE Transactions on Computer Aided Design*, 8(2):171–180, February 1989.

[CW85] Raul Camposano and Raul Weber. Semantik und interne Form von DSL. Interner Bericht 3/85, Forschungszentrum Informatik Karlsruhe, April 1985.

[GR94] Peter Gutberlet and Wolfgang Rosenstiel. Timing Preserving Interface Transformations for the Synthesis of Behavioral VHDL. *EURO-VHDL*, 1994.

[IEE88] IEEE Standard VHDL Language Reference Manual. Technical Report IEEE-1076-1987, The Institute of Electrical and Electronics Engineers, New York, 1988.

[IEE93] IEEE Standard VHDL Language Reference Manual. Technical Report IEEE-1076-1992/B, The Institute of Electrical and Electronics Engineers, New York, 1993.

[KKT+95] T. Kuhn, U. Kebschull, P. Thole, E. Schubert, and W. Rosenstiel. Bibliotheksorientierte Technologieabbildung synthetisierter Datenpfade. *2. GI/ITG Workshop. Anwenderprogrammierbare Schaltungen*, FZI Publikationen Karlsruhe(2):105–111, Juni 1995.

[Leh94] Gunther Lehmann. *Schaltungsdesign mit VHDL*. Franzis-Verlag GmbH, Poing, 1994.

[SMH95] M. Schmidt, U. Möhrke, and P. Herrmann. Synthesis Tools for FPGA-Design. *Institut für Informatik der Universität Leipzig*, Report(15), 1995.

[SMH96] M. Schmidt, U. Möhrke, and P. Herrmann. Verhaltensbeschreibung in der High–Level Synthese. *Institut für Informatik der Universität Leipzig*, Report(3), 1996.